Cerinte:

1. Creati un arbore binar folosind functia (CreateBalanced(int N) – slide 23 – C5). Toate elementele/nodurile din arbore au valori (chei) **distincte si pozitive**. Pentru un set de valori alese de voi descrieti cum se incarca stiva programului (desenati).
2. Parcurgeti arborele in inordine (SRD) – folosind o functie implementata iterativ (idee: similar cu parcurgerea in preordine vezi sliduri 17-20 – C5). Descrieti cum se incarca stiva programului.
3. Implementati o functie pentru eliberarea spatiul ocupat de un arbore binar (iterativ sau recursiv).
   Idee solutie recursiva – traversati arborele in postordine (SDR) si eliberati spatiul pentru nodurile din stanga si dreapta inainte de a elibera spatiul ocupat de radacina. Se elibereaza mai intai spatiul ocupat de nodurile cele mai din stanga jos, dreapta jos.
   Idee solutie iterativa – parcurgeti arborele pe niveluri. Ideea e sa stergeti fiecare nod din coada dupa ce ati adaugat copii acelui nod in coada pentru a fi procesati. Se elibereaza mai intai spatiul ocupat de nodurile cele mai de sus, tinand minte adresele copiilor.
4. Implementati o functie (iterativ sau recursiv) pentru gasirea celui mai apropiat stramos comun a doua noduri (Lowest Common Ancestor - LCA) date prin valoarea lor (toate elementele din arbore au valori distincte).
   LCA a doua noduri a si b este cel mai adanc (de jos nod) care ii are pe a si b ca descendenti.
   LCA pentru 6 si 1 e 8 (Fig. 1).

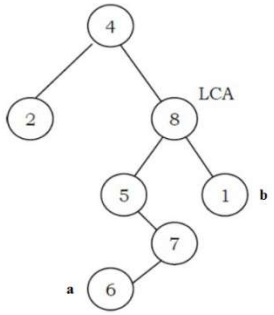

Fig. 1. Arbore binar - LCA

Idee solutie iterativa* - se salveaza caile de la radacina la a si de la radacina la b in 2 vectori. (*Pentru a salva caile oricum am nevoie de o functie recursiva.)
{4, 8, 5, 7}
{4, 8}
Apoi se parcurg vectorii pana cand se gasesc 2 valori diferite (sau nu mai sunt elemente in vreunul dintre cei doi vectori). Ultima valoare egala era cea a LCA.
Ca idee, o functie pentru afisarea stramosilor unui nod cu cheia(valoarea stocata) k se poate implementa asa:

```
int printAllNodeAncestors(Node *root, int k ){
 if (root==NULL) return 0;
//

if (root->val==k) return 1;
if ((root->left!=NULL && printAllNodeAncestors(root->left, k) == 1) ||
            (root->right!=NULL && printAllNodeAncestors(root->right, k) == 1 )) {
        printf("%d", root->val);  /*daca nodul este stramos al nodului cu cheia k, il afisez*/
        return 1;
      }
//
 return 0;
}
```

Aceasta functie trebuie modificata astfel incat, intr-un vector sa adaug toate nodurile. Pentru a face acest lucru, orice nod e adaugat in vector la final, apoi se verifica daca e stramos, si daca nu este, este scos din vector.
Deoarece arborele stocheaza valori pozitive, putem sa creem un vector cu N elemente (unde N este numarul de noduri) sa initializam toate elementele cu -1; adaugarea presupune stocarea cheii nodului parcurs, iar eliminarea presupune ca "elementul de sters" sa redevina -1.
Se poate folosi o coada in loc de vector.

Idee solutie complet recursiva – trebuie gasit nodul din arbore care il are pe a intr-un subarbore si pe b in celalalt subarbore.
Se foloseste parcurgerea in postordine si se verifica la fiecare pas:
- daca nodul investigat e NULL se returneaza NULL
- daca nodul investigat are cheia a sau b se returneaza adresa acelui nod (este de interes ca l-am gasit). OBS: Daca o cheie este stramos al celeilalte, atunci nodul care o contine e LCA
- daca nodul nu e nici NULL si nu contine cheia a sau b atunci se cauta in subarborele stang si, respectiv, drept dupa cele 2 chei; fiecare cautare isi stocheaza rezultatul intr-o variabila locala astfel: daca se gaseste vreo cheie in subarborele stang se salveaza adresa nodului in lca_l si daca se gaseste in cel drept in lca_r
- daca in urma cautarii lca_l si lca_r sunt nenule inseamna ca a e prezent intr-un subarbore si b in celalalt - atunci nodul curent e LCA => se returneaza acest nod.
- altfel, LCA se gaseste in subarborele stang sau drept (lca_l sau lca_r) pe care il returnam astfel incat cautarea sa continue.